

# Prolog and Cut

Mattox Beckman

beckman@iit.edu

Illinois Institute of Technology

## Objectives

## §0 Objectives

Prolog's greatest strength is its ability to backtrack to find alternative solutions. If not controlled, it can also be its greatest weakness. In this lecture we will go over the cut operator, which gives a solution to this problem.

- Know what the cut operator is and what it does.
- Know how to use the cut operator to assert failure.
- Know how to use the cut operator to stop recursion.

## Prolog and Cut Backtracking

## §1 Backtracking

```

1 color(red).
2 color(blue).
3 car(honda).
4 car(ford).
5 car(toyota).
6
7 ?- color(A), car(B).
```

```

1 A = red
2 B = honda ;
3 A = red
4 B = ford ;
5 A = red
6 B = toyota ;
7 A = blue
8 B = honda ;
9 A = blue
10 B = ford ;
11 A = blue
12 B = toyota ;
```

Come and see the backtracking inherent in the system!

## Prolog and Cut The Cut operator

## §1 Backtracking

- The Cut operator (!) stops backtracking.
- It is considered a goal that always succeeds.

```

1 ?- color(A), !, car(B).
2
3 A = red
4 B = honda ;
5 A = red
6 B = ford ;
7 A = red
8 B = toyota ;
9 No
```

Once a cut is activated, the clause we are trying to satisfy is committed to that choice.

```
1 color(red).
2 color(green) :- !.
3 color(blue).
4
5 ?- color(X).
6 X = red ;
7 X = green ;
8 No
```

Once X was set to green, the cut operator forces us to stay with green or else color should fail completely. **Question:** Can color(blue) ever be matched?

You can add a constraint to the second clause....

```
1 fact(0,1).
2 fact(N,X) :- N > 0, M is N-1, fact(M,Y), X is Y * N.
```

Or you can add a cut to the first clause.

```
1 fact(0,1) :- !.
2 fact(N,X) :- M is N-1, fact(M,Y), X is Y * N.
```

Now it will work:

```
1 ?- fact(5,N).
2 N = 120 ;
3 No
```

```
1 fact(0,1).
2 fact(N,X) :- M is N-1, fact(M,Y), X is Y * N.
3
4 ?- fact(5,N).
5
6 N = 120 ;
7 ERROR: Out of local stack
```

🚫 What happened here?

Suppose you run the campus observatory. You want to allow certain people to use the telescope. They have to be a student, a faculty member, or a member of the astronomy club. And they also need to have been trained on the telescope.

**students** anna, beth, cindy, david

**faculty** ernest, frank, gloria

**astronomy club** anna, frank, harry

**trained** anna, harry

**Can frank use the telescope?**

§3 Efficiency

```

1 telescope(X) :- (student(X); faculty(X); club(X)),
2                 trained(X).

```

- frank is a faculty, and also a member of the club.
- But, frank doesn't have any training.

What will `telescope(frank). do?`

**Now we cut...**

§3 Efficiency

```

1 telescope(X) :- (student(X); faculty(X); club(X)),
2                 !,
3                 trained(X).
4
5 ?- telescope(X).
6 X = anna ;
7 No

```

- Oops. Now we've dissed harry.
- But at least we don't spend a lot of time when we ask if frank can use the telescope....
- Moral: cut will limit your choices to only one answer.

**Who can use the telescope?**

§3 Efficiency

```

1 telescope(X) :- (student(X); faculty(X); club(X)),
2                 trained(X).
3
4 ?- telescope(X).
5 X = anna ;
6 X = anna ;
7 X = harry ;

```

Since anna is a student *and* a member of the club, she gets listed twice.

**Inducing failure.**

§3 Efficiency

- We also have a predicate called `fail`, which, well, always fails.
- Suppose anna has her telescope privileges revoked...

```

1 telescope(anna) :- fail.
2 telescope(X) :- (student(X); faculty(X); club(X)),
3                 trained(X).
4
5 ? telescope(anna).
6 Yes.

```

This is less than what we hoped for.

```
1 telescope(anna) :- !, fail.  
2 telescope(X) :- (student(X); faculty(X); club(X)),  
3                 trained(X).  
4  
5 ? telescope(anna).  
6 No  
7 ?- telescope(harry).  
8 Yes  
9 ?- telescope(X).  
10 No
```

But cut and fail will work.

1. Write a predicate `between(X,Y,Z)` which is true when `Y` is a point between `X` and `Z`.
2. Write a predicate `grandfatherof(X,Y)` which is true when `X` is a grandfather of `Y`. You might want to write another predicate first.
3. Write a predicate `flatten(X,Y)` that is true when `Y` is a flattened version of `X`. E.g.

```
1 ?- flatten([x,3,[a,b],3,[[3]]],[x,3,a,b,3,3]).  
2 Yes
```

Hint: there is a predicate called `is_list`, which is true when its argument is a list.

- Cut can stop searches that you already know will be useless.
- Cut can make queries more efficient.
- But, cut can make queries do strange things. Use with care.

Aside: you can define `not` (actually, it's built in) this way:

```
1 not(X) :- call(X), !, fail.  
2 not(X).
```

This predicate can fix the telescope problems.

```
1 between(X,Y,Z) :- pathfrom(X,Y), pathfrom(Y,Z).  
2  
3 parentof(X,Y) :- fatherof(X,Y); motherof(X,Y).  
4  
5 grandfatherof(X,Y) :- fatherof(X,Z), parentof(Z,Y).
```

```

1 myflatten([H|T],X) :- is_list(H), append(H,T,R),
2                       myflatten(R,X).
3 myflatten([H|T],[H|X]) :- myflatten(T,X).
4 myflatten([],[]).
5
6 ?- myflatten([[2,3],[3,4],[5,6],4],3,X).
7 X = [2, 3, 3, 4, 5, 6, 4, 3] ;
8 X = [2, 3, 3, 4, [5, 6], 4, 3] ;
9 X = [2, 3, [3, 4, [5, 6], 4], 3] ;
10 X = [[2, 3], 3, 4, 5, 6, 4, 3] ;
11 X = [[2, 3], 3, 4, [5, 6], 4, 3] ;
12 X = [[2, 3], [3, 4, [5, 6], 4], 3] ;
13 No

```

```

1 myflatten([H|T],X) :- is_list(H), !, append(H,T,R),
2                       myflatten(R,X).
3 myflatten([H|T],[H|X]) :- myflatten(T,X).
4 myflatten([],[]).

```